

The background of the slide features a complex, abstract molecular structure composed of interconnected hexagons and lines, rendered in various shades of blue. Some vertices of the hexagons are highlighted with bright blue, glowing points, giving the impression of a network or a chemical structure. The overall aesthetic is high-tech and digital.

UMA CLASSE **JAVA**

O modificador de acesso **public**, torna essa classe acessível a outras classes.

Isso significa que outras classes, independentemente de estarem no mesmo pacote ou em pacotes diferentes, podem criar instâncias dessa classe.

```
public class Cadeira {  
  
}
```

O modificador de acesso **public**, torna essa classe acessível a outras classes.

Isso significa que outras classes, independentemente de estarem no mesmo pacote ou em pacotes diferentes, podem criar instâncias dessa classe.

A classe tem que ser salva em um arquivo chamado Cadeira.java

```
public class Cadeira {  
  
}
```

```
public class Cadeira {  
    String posicao;  
    boolean ocupado;  
  
    void sentar() {  
    }  
    void levantar() {  
    }  
    void virar() {  
    }  
    String getPosicao() {  
    }  
}
```



```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public void sentar() {  
    }  
    public void levantar() {  
    }  
    public void virar() {  
    }  
    public String getPosicao() {  
    }  
}
```

Cada membro também tem o seu
controle de acesso

```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public void sentar() {  
    }  
    public void levantar() {  
    }  
    public void virar() {  
    }  
    public String getPosicao() {  
    }  
}
```

Cada membro também tem o seu controle de acesso

private, significa que esse membro só pode ser acessado dentro da própria classe onde foi definido. Isso cria um encapsulamento dos detalhes internos da classe, impedindo o acesso direto a esses membros por outras classes.

```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public void sentar() {  
    }  
    public void levantar() {  
    }  
    public void virar() {  
    }  
    public String getPosicao() {  
    }  
}
```



Cada membro também tem o seu
controle de acesso

private, significa que esse membro
só pode ser acessado dentro da
própria classe onde foi definido.
Isso cria um encapsulamento dos
detalhes internos da classe,
impedindo o acesso direto a esses
membros por outras classes.

```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public void sentar() {  
    }  
    public void levantar() {  
    }  
    public void virar() {  
    }  
    public String getPosicao() {  
    }  
}
```

Cada membro também tem o seu controle de acesso

os membros declarados como **public** são acessíveis de fora da classe em que são definidos. Isso significa que eles podem ser acessados e utilizados por outras classes, independentemente de estarem no mesmo pacote ou em pacotes diferentes.

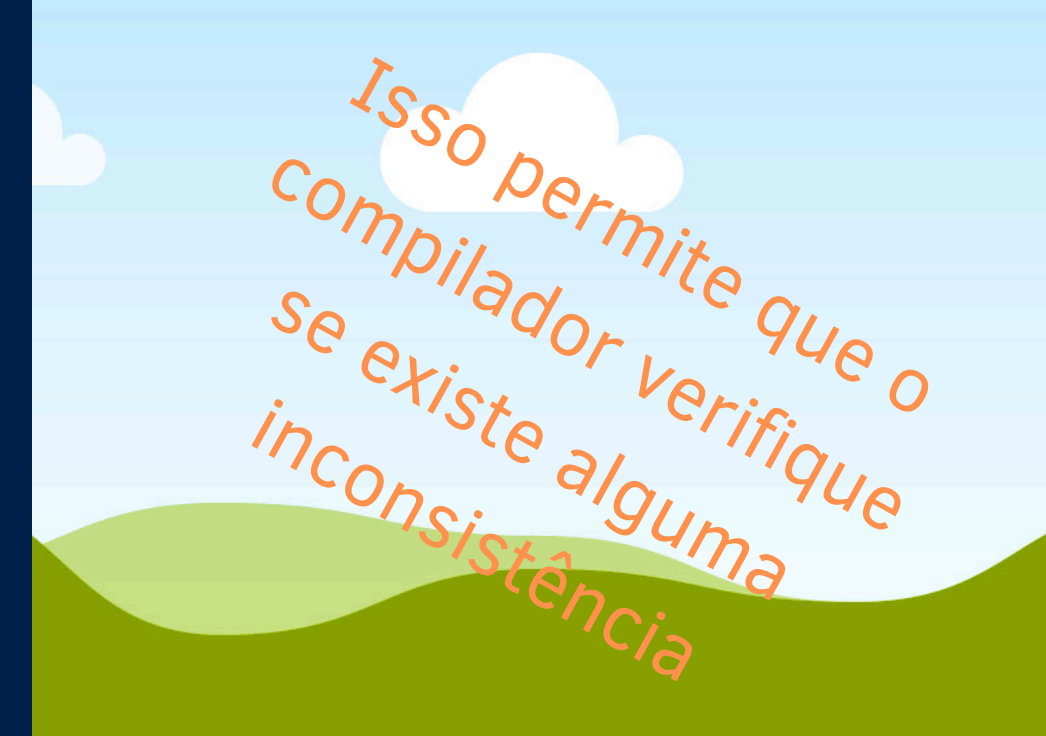
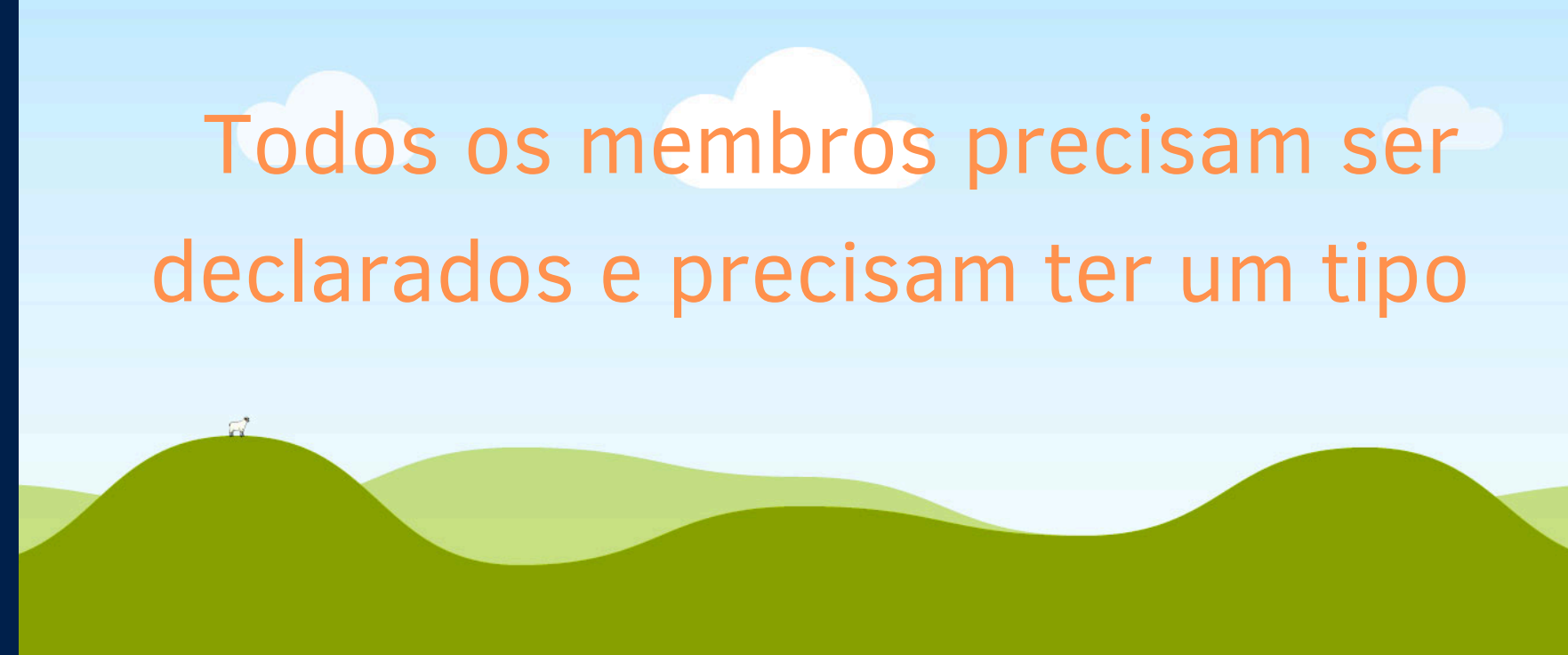

```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public Cadeira() {  
    }  
    public Cadeira(String p,  
                    boolean oc) {  
    }  
    public void sentar() {  
    }  
    public void levantar() {  
    }  
}
```

Construtor é um método que tem o mesmo nome da classe.

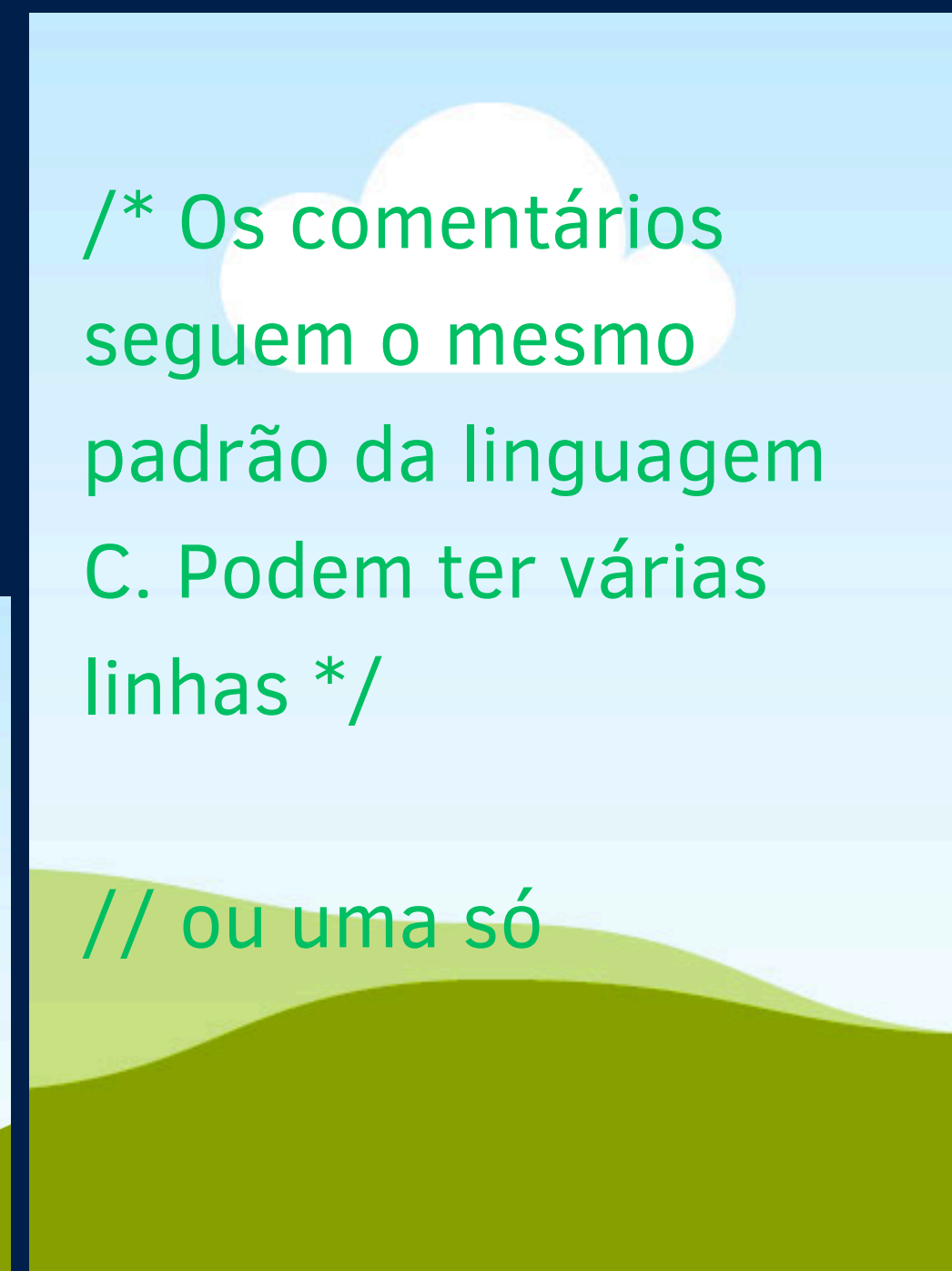
Não tem tipo declarado.

Podemos ter mais do que um construtor.

```
c1 = new Cadeira();  
c2 = new Cadeira("Em pé", false);
```



Observações





Semelhantes aos tipos da
linguagem C

Tipos

Tipo	Tamanho
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
short	2 bytes
byte	1 byte
boolean	1 bit



Semelhantes aos tipos da
linguagem C

Condições de comandos como
if e while aceitam unicamente
valores booleanos

Tipos

Tipo	Tamanho
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
short	2 bytes
byte	1 byte
boolean	1 bit



Operadores

Tipo do operador	Lista de operadores
Sufixal	expr++ expr--
Prefixal	++expr --expr +expr -expr ~ !
Multiplicativos	* / %
Aditivos	+ -
Shift binário	<< >> >>>
Comparativos	< > <= >= instanceof
Igualdade	== !=
Bit-a-bit E	&
Bit-a-bit XOU	^
Bit-a-bit OU	
Lógico E	&&
Lógico OU	
Ternário	? :
Atribuição	= += -= *= /= %= &= ^= = <<= >>= >>>=

x++; 8 + y++;

--x; 8 + --y;

x = y >> 2;

a <= b

a == b a != b

x = y & 0b101

a > b && b <= c

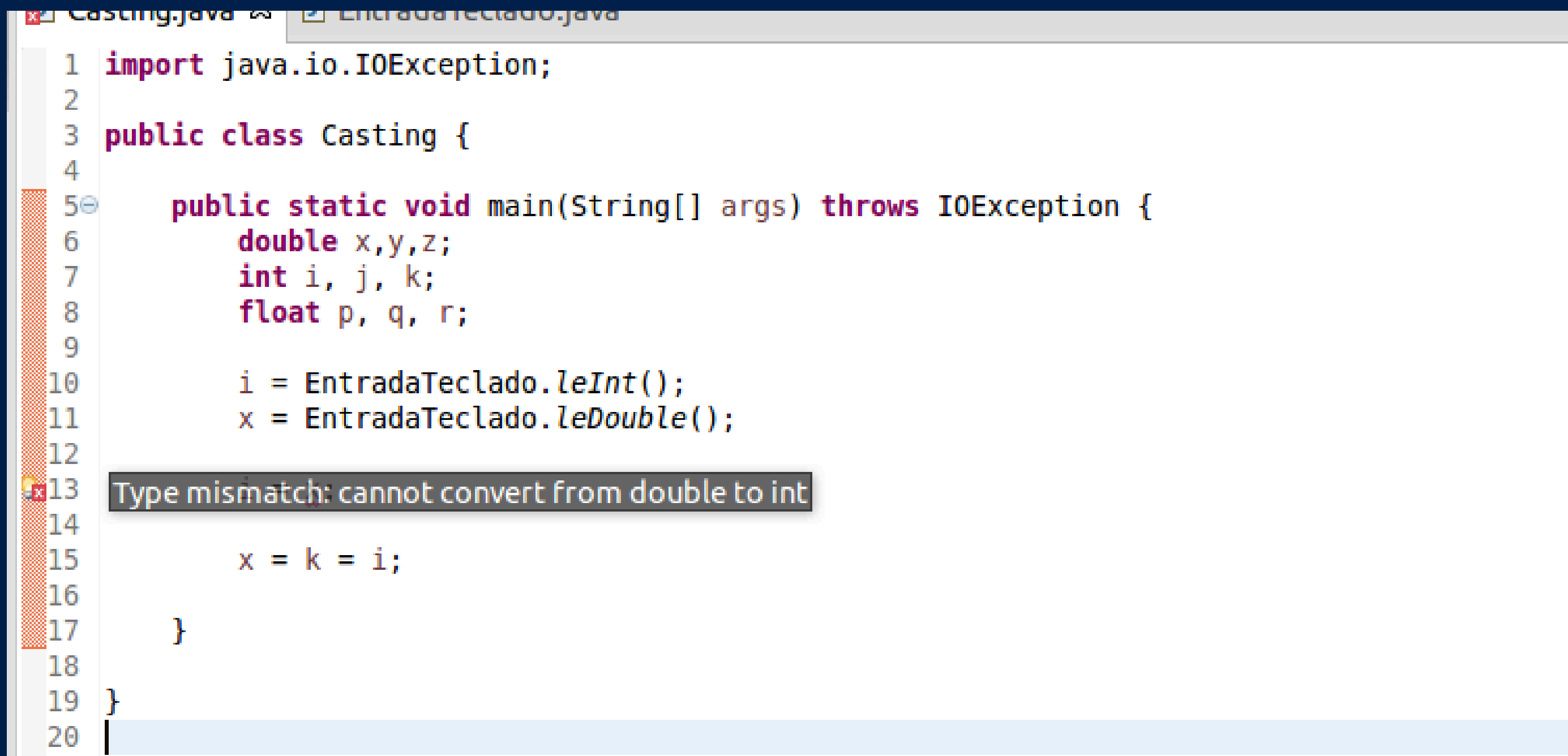
x = a > b ? b : b;

X += 10;

Casting

- 1 Em C é possível fazer o casting de tipos
- 2 Em Java, é obrigatório em muitos casos fazer o casting
- 3 Quando há possibilidade de se perder informação o casting é requerido
- 4 Compilador vai avisar

Erro de conversão de tipo



The screenshot shows a Java IDE with two tabs: 'Casting.java' and 'EntradaTeclado.java'. The 'Casting.java' tab is active, displaying the following code:

```
1 import java.io.IOException;
2
3 public class Casting {
4
5     public static void main(String[] args) throws IOException {
6         double x,y,z;
7         int i, j, k;
8         float p, q, r;
9
10        i = EntradaTeclado.leInt();
11        x = EntradaTeclado.leDouble();
12
13        x = k = i;
14
15    }
16
17 }
18
19
20
```

A red squiggly line is under the variable 'x' on line 13. A tooltip with a red error icon points to this line, displaying the message: "Type mismatch: cannot convert from double to int".

Tabela de conversão

PARA:	byte	short	char	int	long	float	double
DE:							
byte	----	<i>Impl.</i>	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
short	(byte)	----	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
char	(byte)	(short)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
int	(byte)	(short)	(char)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
long	(byte)	(short)	(char)	(int)	----	<i>Impl.</i>	<i>Impl.</i>
float	(byte)	(short)	(char)	(int)	(long)	----	<i>Impl.</i>

Declaração de variáveis

- Variáveis podem ser declaradas na hora que forem usadas
- Vale a mesma regra de escopo
- Por exemplo uma variável declarada dentro de um if vale apenas naquele escopoEscreva seu ponto de agenda
- Parâmetros são variáveis locais

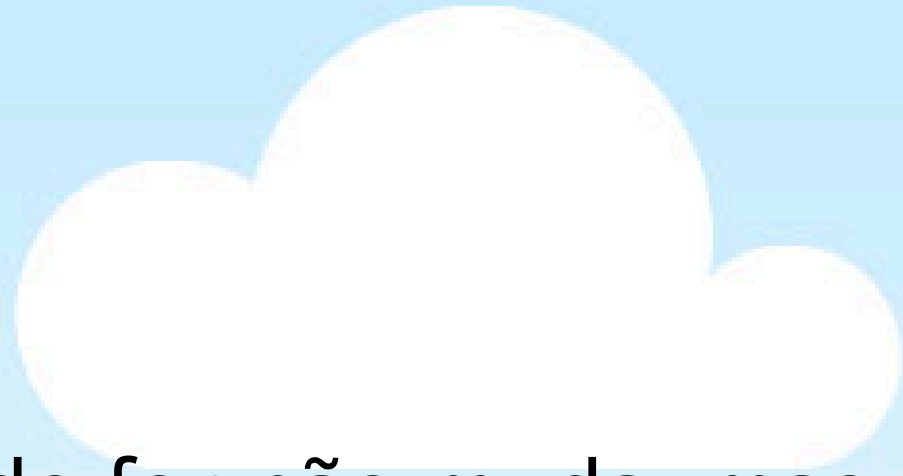
Comandos de seleção

1

```
if (expressão booleana)
{
    comando 1;
    comando 2;
}
else {
    comando 3;
    comando 4;
}
```

2

```
switch (s) {
    case "abc":
        b = 10;
        break;
    case "cde":
        c = 20;
        break;
    default:
        b = 0;
}
```



Comando for não muda, mas podemos declarar a variável que será usada como controle.

Também existe for para percorrer arrays, Strings e outros

Comandos while não mudam

Repetição

```
for (int i = 0; i < 10; i++) {  
}
```

```
for (int k : v) {  
}
```

```
while ( i < 10 ) {  
}
```

```
do {  
} while ( i < 10)
```

Break e continue

Tudo igual, mas podemos ter um rótulo que indica qual comando quebrar ou continuar.



```
boolean achou = false;
label1:
for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
        if (t[i][j] == 0) {
            achou = true;
            break label1;
        }
```

Exceções

```
fscanf(arq, "%d", &k);
```

Como saber se um erro ocorreu?

```
if (fscanf(arq, "%d", &k) != 1)
{
    printf("Erro na leitura do arquivo");
    return -1;
}
```

Sinalização que um erro ocorreu

Exceções

```
InputStreamReader in = new InputStreamReader(System.in);  
BufferedReader br = new BufferedReader(in);  
String arq = br.readLine();  
FileInputStream fis = new FileInputStream(arq);  
InputStreamReader ins = new InputStreamReader(System.in);  
BufferedReader brs = new BufferedReader(ins);
```

Exception in thread "main" java.io.FileNotFoundException: aa (Arquivo ou diretório inexistente)
at java.base/java.io.FileInputStream.open0(Native Method)
at java.base/java.io.FileInputStream.open(FileInputStream.java:211)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:108)
at Except.main(Except.java:12)

Exceções

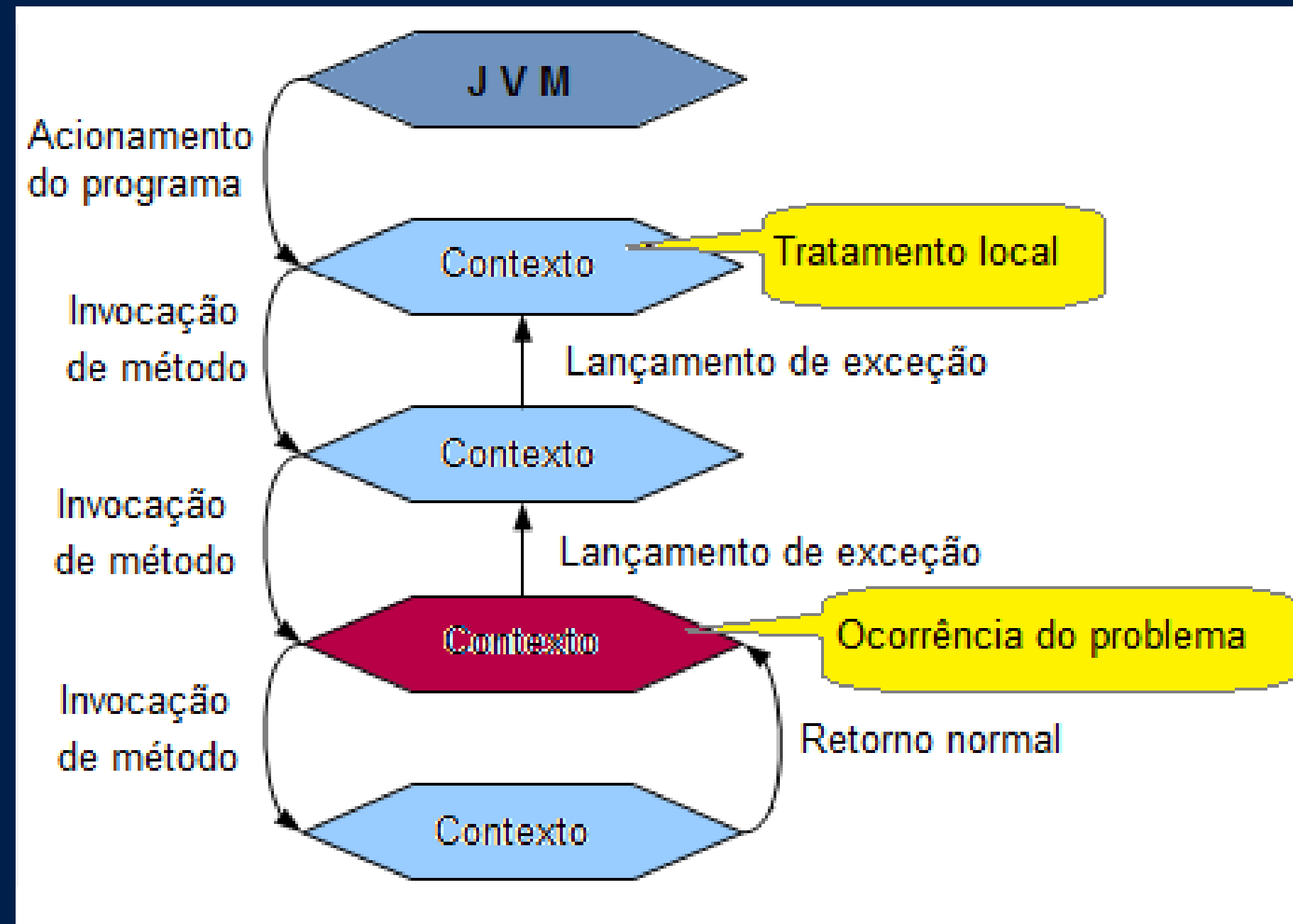
```
try {  
    InputStreamReader in = new InputStreamReader(System.in);  
    BufferedReader br = new BufferedReader(in);  
    String arq = br.readLine();  
    FileInputStream fis = new FileInputStream(arq);  
    InputStreamReader ins = new InputStreamReader(System.in);  
    BufferedReader brs = new BufferedReader(ins);  
}  
catch (Exception e) {  
    System.out.println("Erro ao ler arquivo");  
    ;  
}
```

Qualquer exceção dentro do bloco é tratada.

Evita que a exceção seja propagada para quem chamo o método em questão.

Se a exceção for propagada, quem fez a chamada ainda pode tratar a exceção.

Tratamento de exceções





Cadeira.java

```
public class Cadeira {  
    private String posicao;  
    private boolean ocupado;  
  
    public Cadeira() {  
        ocupado = false;  
        posicao = "Em pé";  
    }  
  
    public Cadeira(String p,  
                    boolean oc) {  
        posicao = p;  
        ocupado = oc;  
    }  
  
    public void sentar() {  
        if ( (! ocupado) && posicao.equals("Em pé") )  
            ocupado = true;  
    }  
  
    public void levantar() {  
        ocupado = false;  
    }  
  
    public void virar() {  
        if (posicao.equals("Em pé")) {  
            posicao = "Invertida";  
            ocupado = false;  
        }  
        else  
            posicao = "Em pé";  
    }  
  
    public String getPosicao() {  
        return posicao;  
    }  
}
```

Cadeira.java

```
public class Cadeira {
    private String posicao;
    private boolean ocupado;

    public Cadeira() {
        ocupado = false;
        posicao = "Em pé";
    }

    public Cadeira(String p,
                    boolean oc) {
        posicao = p;
        ocupado = oc;
    }

    public void sentar() {
        if ( (! ocupado) && posicao.equals("Em pé") )
            ocupado = true;
    }

    public void levantar() {
        ocupado = false;
    }

    public void virar() {
        if (posicao.equals("Em pé")) {
            posicao = "Invertida";
            ocupado = false;
        }
        else
            posicao = "Em pé";
    }

    public String getPosicao() {
        return posicao;
    }
}
```

```
static public void main(String args[]) {
    Cadeira c1 = new Cadeira();
    Cadeira c2 = new Cadeira("Invertida", false);
    c1.sentar();
    c2.virar();
    System.out.println(c1.getPosicao());
    System.out.println(c2.getPosicao());
}
```

Main.java

```
public class Main {  
  
    static public void main(String args[]) {  
        Cadeira c1 = new Cadeira();  
        Cadeira c2 = new Cadeira("Invertida", false);  
        c1.sentar();  
        c2.virar();  
        System.out.println(c1.getPosicao());  
        System.out.println(c2.getPosicao());  
    }  
}
```

Escreva um programa principal que permita que o usuário movimente as peças até chegar na solução



Escreva um programa principal que ache uma sequência de movimentos que leve à solução. (Busca em largura/profundidade)



Inclua um construtor no qual a posição inicial do tabuleiro possa ser fornecida

Arrays

```
private int[] table; // define uma variável
```

```
table = new int[16];
```

```
tamanho = table.length;
```

```
for (int i= 0; i < tamanho; i++) {
```

```
    table[i] = i;
```

```
}
```

Spoiler



```
private int[][] table;
```

```
table = new int[4][4];  
linhas = table.length;
```

```
int k = 0;
```

```
for (int i= 0; i < linhas; i++) {  
    for (int j = 0; j <  table[i].length; j++) {  
        table[i][j] = k++;  
    }  
}
```

String

```
System.out.println("Digite uma opção (U/D/L/R) ou Q para sair: ");
String r;
r = "Digite uma opção (U/D/L/R) ou Q para sair: ";

String h = "Digite uma opção (U/D/L/R) ou Q para sair: ";

String c = "Digite uma opção" + " (U/D/L/R) ou Q para sair: ";

int k = c.length();
String s = h.substring(4,10);
```